Minimal Logging in C++ 20
Koen Poppe
28/06/2022
Existing approaches

01 Qt

• Streaming operators
  ▪ Automatic newlines

• Severities
  ▪ qInfo()
  ▪ qDebug()
  ▪ qWarning()

• Categories

• Output configured at runtime
  ▪ Filtering (category + severity)
  ▪ Output format
Existing approaches
02 iostream

- Anything you want
  - i.e., use std::cout

- Could use macros to avoid repetition
  - __FILE__
  - __FUNCTION__
  - __LINE__

```cpp
#define LOG(message) std::cout << __FILE__ << "(" << __LINE__ << ")" << __FUNCTION__ << " : " message "\n";
```
- I do not like macros …
Existing approaches
03 SourceLocation

• **C++ 20** `std::source_location`
  ▪ `#include <source_location>`

```cpp
void log(const std::string_view message, const std::source_location location = std::source_location::current()) {
    std::cout << location.file_name() << "(" << location.line() << ")" << location.function_name() << "\": " << message << '\n';
}
```

• Prevent macros using default argument
  ▪ Requires compiler support
Existing approaches

• Why code names?

• Also: why log plain text*
  ▪ Hard to process in automated systems

• How about the debugger?

*: There is no such thing as plain text Plain Text - Dylan Beattie - NDC Oslo 2021
Intermezzo 1: Debugger

- Knows this already...

- Stack frames
  - Function call = new stack frame
  - Contains return address

- “Stack walking”
  - Reverse walk from current

- Look up addresses in debugging info
Existing approaches

- Include debugging symbols?
  - No!
    - Increases binary size significantly
    - Intellectual property
      - void writePropietaryFileFormat(…)
    - Security risk
      - SuperSecretSoftwareProtection.cpp
      - bool checkLicenseKey(…)

<table>
<thead>
<tr>
<th></th>
<th>Without</th>
<th>With</th>
</tr>
</thead>
<tbody>
<tr>
<td>No logging</td>
<td>37 KB</td>
<td>69 KB</td>
</tr>
<tr>
<td>Qt</td>
<td>42 KB</td>
<td>419 KB</td>
</tr>
<tr>
<td>iostream</td>
<td>40 KB</td>
<td>94 KB</td>
</tr>
<tr>
<td>SourceLocation</td>
<td>40 KB</td>
<td>105 KB</td>
</tr>
</tbody>
</table>
Intermezzo 2: Crash dumps

- Crash dump contains stack
  - Use “Stack walking” to determine location
Symbols for logging?

• Using symbols
  ▪ Translate back to
    • Function
    • File/line
    • ...

• What else is required?
  ▪ The instruction pointer
    • Special register, cheap to obtain
Minimal Logging
Unit tests

• Live coding is a risk
  ▪ Unit tests as safety net

• Run tests for non-compiling code?
  ▪ **C++20** Concepts to the rescue!

```cpp
if constexpr (requires(Logger l) {
    l.rocketscience();
})
{
    Logger logger;
    logger.rocketscience();
    ...
}
else
{
    QSKIP("Does not compile");
}
```
Minimal Logging
Phase 1/3

- Trace the instruction pointer
  - Retrieve it using inline assembly
  - Write it into a buffer

- Test translation back into
  - Function
  - File & line
    - This does not work ~ compiler support std::source_location::current
    - Could be done using macros
Minimal Logging Framework in C++ 20 - Koen Poppe - BeC++ - 28/06/2022

Phase 1/3: remarks

• (Reserved field for next phases)

• `uintptr_t`
  ▪ Representation of an address

• No string literals in the binary
  ▪ Actually using the symbols file

• C++11 `__attribute__((always_inline))`
  ▪ Impact inlining (still requires optimized build)
Minimal Logging
Phase 2/3

• Add time information
  ▪ `std::chrono`
Minimal Logging Framework in C++ 20 - Koen Poppe - BeC++ - 28/06/2022

Phase 2/3: remarks

- Choose a clock
  - Clock dependent types to not loose data

- Type safe handling of time
  - Was it sleep(seconds) or sleep(milliseconds)?
  - See also Boost::Units
Minimal Logging
Phase 3/3

- Argument
- Arguments
Minimal Logging
Phase 3/3: remarks

• Use std::tuple like structure to represent arguments
  ▪ Use custom packed type to avoid padding

• How to decode?
  ▪ Trace the trace method

• C++17 std::any to represent data
  ▪ Type safe union

• __attribute__((used)) to avoid function collapsing
  ▪ LLVM specific?
Open points

- Resolving file:line
  - Any ideas?

- Symbol versioning
  - Link with source code revision

- Shared libraries
  - All log to the same buffer
  - Special handling for symbols

- Thread safety
  - Lock free buffer

- Handling of fixed literals
  - Why log constant information?

- Include in crash dumps
  - Already the case (stack allocated)
Minimal Logging Framework in C++ 20

Questions?

1. How about strings?
2. How does this compare to spdlog
   (gabime/spdlog: Fast C++ logging library. (github.com))
Minimal Logging Framework in C++ 20

inspired by Expertise